TOWARDS COMPREHENSIVE RESILIENCE FOR SCIENTIFIC APPLICATIONS

THE RELIABILITY CHALLENGE Future HPC systems are expected to be significantly less reliable than past systems for three reasons. First, as the circuit feature sizes grow smaller, they become physically less able to maintain their signals (e.g. inter-wire distance shrinks to just a few atoms) (1). Second, power-constrained Exascale systems will need to be built from processors where less power is available to maintain consistent operation (2). Finally, the HPC market's small size means that Exascale systems will be built from parts designed for larger, more profitable markets.

Together, these trends mean that Exascale system designers will find it very challenging to build a fully reliable system. Components from the server market will be as reliable as Oracle or Google need them to be but will not be as power efficient as required for Exascale. Components used by embedded devices (e.g. cell phones) will achieve the required power efficiency but not reliability. Fundamentally, no other market has the same power constraints and large scale as HPC and thus, components from such markets will provide inadequate power/reliability tradeoffs. Since funding constraints cap the operating cost of an Exascale system to $20-$30m/year (i.e. 20-30 MW), Exascale designs must sacrifice reliability to deliver an operational system in a timely manner.

THE CHALLENGE **The likely constraints on the reliability of Exascle systems make it critical to fund and carry out a comprehensive research program to (i) design scientific and numerical algorithms that are resilient to a wide range of faults and (ii) evaluate their effectiveness in many different possible system scenarios. Although this challenge is at least as difficult as achieving high Exascale performance, the lack of dedicated attention to this area over the past decades leaves us decades behind and no time to lose.**

FAULT MODEL The first key challenge in any software resilience strategy is to clearly identify the types of faults software must be resilient to. Each fault type can be handled by a different class of solutions and some types may be inherently easier to handle for different types of applications. Knowing the relationship between fault type, fault frequency and the cost of handling faults in software informs system designers and application developers regarding the flexibility realistically available for Exascale system designs. Specifically, the following types of faults must be considered:

- Fail-stop: Components (e.g. compute nodes or network switches) cease operation without corrupting any system or application state. Can be managed by generic checkpointing (3) (4) (5) (6) or replication (7) or application-specific techniques that recompute lost application state (8).
- Silent Data Corruption: Introduce errors into application state or communication (e.g. transient voltage variation). Can be managed by generic replication (7) or application-specific techniques that check algorithmic invariants (9) (10) (11) and use them to fix the corruption (12) (13).
- Detected Data Corruption: Errors may occur in system or application state but are quickly detected by hardware or OS mechanisms (14) (15). Can be managed more cheaply than above by localized rollback techniques such as message logging (16) or containment domains (17).
- Performance Variation: The performance of various components degrades non-deterministically. Can be detected via statistical techniques (18) (19) and managed by latency-tolerant algorithms.

A design's productivity is determined by the performance and code development costs of handling each fault type and the cost of constraining hardware faults to a given type. We thus must develop methods to evaluate algorithm behavior given any distribution of faults that a system may exhibit.

ALGORITHM DESIGN Although there exist generic techniques to detect and tolerate most types of faults (e.g. checkpointing or replication), they can be expensive in both performance and power efficiency. It is thus critical to develop new scientific and numerical algorithms or extensions of prior algorithms that are more resilient to the fault types listed above. The primary challenge of this work is to create and exploit key invariants of each algorithm to efficiently detect and correct errors. For example, in the context of linear algebra computations (e.g. matrix-matrix multiplication $AB = C$) it is possible to detect and correct errors by encoding the inputs using a linear error correcting code (10) and checking whether the encoding is preserved by the computation (e.g. $x^T ABy \stackrel{?}{=} x^T Cy$). The same encoding can be used to correct a limited number of errors and as we have shown, can be extended to sparse linear operations (9) (12) (11). Further, we have shown that algorithms such as the Algebraic Multi-Grid are naturally resilient to numeric errors and require replication of pointers to become fully resilient (13).

Although prior research on resilient algorithms has demonstrated that many individual algorithms can be made resilient, the fact is that making any new algorithm resilient is very challenging. In particular, data corruptions may push iterative algorithms outside their convergence region. Worse, they may corrupt key data structures, causing the algorithm to produce erroneous results. Further, real simulations, which combine many code modules are too complex for traditional algorithmic resilience techniques based on simple properties such as linearity. This means that we need aggressive research on new classes of algorithmic resilience techniques that address more generic application classes. For example, polynomial computations are significantly more general than linear functions but have sufficient structure to enable powerful error detection and correction techniques. Further, many non-linear algorithms can be locally approximated via linear functions or polynomials. Similarly, since the states of most simulations only show significant variation around shocks, it should be possible to efficiently detect and correct errors within them by using full replication around shocks and interpolation in smooth regions.

Finally, since hardware faults may manifest themselves as unpredictable slowdowns of various system components (e.g. frequent memory error corrections) it is imperative to design algorithms that perform well even if some computations are transiently slower than usual (20).

COMPOSITIONAL RELIABILITY Even as individual application modules such as solvers are made resilient to faults, it is critical to ensure that applications composed of such modules are also resilient. For instance, although one module may tolerate corruptions in individual data elements (e.g. AMG tolerates high-frequency error), they may not be as successful when small errors from other modules propagate to their inputs. Further, while one module may tolerate computation delays, two composed modules may be highly sensitive to delays in each other's results.

SUMMARY Limits of circuit design, power efficiency and a small market make it likely that Exascale systems will be unreliable. The grand challenge of Exascale resilience is thus high productivity and cost effectiveness. Although it is possible to make any system or application resilient to any type of fault, ensuring low cost comprehensive resilience requires research on hybrid solutions that combine algorithmic, system software and hardware resilience techniques. These solutions must be evaluated on realistic representations of faults to quantify the range of system designs for which they are effective. While our work has considered multiple aspects of this problem (21), more foundational research with real applications and systems is needed.

BIBLIOGRAPHY

1. **ITRS.** *International Technology Roadmap for Semiconductors.* 2012.

2. **Peter Kogge, ed.** *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems.* Defense Advanced Research Projects Agency. 2008.

3. *Compiler-Enhanced Incremental Checkpointing for OpenMP Applications.* **Greg Bronevetsky, Daniel Marques, Keshav Pingali, Radu Rugina, Sally A. McKee.** 2009. IEEE International Parallel & Distributed Processing Symposium .

4. *Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System.* **Adam Moody, Greg Bronevetsky, Kathryn Mohror and Bronis R. de Supinski.** 2010. IEEE/ACM Supercomputing Conference (SC).

5. *Application-level Checkpointing for OpenMP Programs.* **Greg Bronevetsky, Keshav Pingali, Paul Stodghill.** 2006. International Conference on Supercomputing.

6. *Automated Application-level Checkpointing of MPI Programs.* **Greg Bronevetsky, Daniel Marques, Keshav Pingali, and Paul Stodghill.** 2003. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming.

7. *Evaluating the Viability of Process Replication Reliability for Exascale Systems.* **Kurt Ferreira, Rolf Riesen, Patrick G. Bridges, Dorian Arnold, Steraley, James H. Laros III, Ron Oldfield, Kevin Pedretti, Ron Brightwell.** 2011. ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis.

8. *Algorithm-based Checkpoint-free Fault Tolerance for Parallel Matrix Computations on Volatile Resources.* **Zizhong Chen, Jack Dongarra.** 2006. International Conference on Parallel and Distributed Processing.

9. *Algorithmic Approaches to Low Overhead Fault Detection for Sparse Linear Algebra.* **Joseph Sloan, Rakesh Kumar and Greg Bronevetsky.** 2012. International Conference on Dependable Systems and Networks (DSN).

10. *Algorithm-Based Fault Terance for Matrix Operations.* **Abraham, Kuang-Hua Huang and J.A.** 6, 1984, IEEE Transactions on Computers, Vol. 33.

11. *Soft Error Vulnerability of Iterative Linear Algebra Methods.* **Greg Bronevetsky, Bronis de Supinski.** 2008. International Conference on Supercomputing.

12. *An Algorithmic Approach to Error Localization and Partial Recomputation for Low-Overhead Fault Tolerance on Parallel Systems.* **Joseph Sloan, Greg Bronevetsky Rakesh Kumar.** 2013, International Conference on Dependable Systems and Networks.

13. *Fault Resilience of the Algebraic Multi-grid Solver.* **Marc Casas, Bronis R. de Supinski, Greg Bronevetsky, Martin Schulz.** 2012.

14. *Low-cost Program-level Detectors for Reducing Silent Data Corruptions.* **Siva Kumar Sastry Hari, Sarita V. Adve, Helia Naeimi.** 2012. International Conference on Dependable Systems and Networks.

15. *Robust System Design with Built-In Soft Error Resilience.* **S. Mitra, N. Seifert, M. Zhang, Q. Shi, K.S. Kim,.** 2, 2005, IEEE Computer, Vol. 38, pp. 43-52.

16. *Dynamic Load Balance for Optimized Message Logging in Fault Tolerant HPC Applications.* **Esteban Meneses, Laxmikant V. Kalé, Greg Bronevetsky.** 2011. IEEE International Conference on Cluster Computing.

17. *Containment Domains: A Scalable, Efficient, and Flexible Resilience Scheme for Exascale Systems.* **Jinsuk Chung, Ikhwan Lee, Michael Sullivan, Jee Ho Ryoo, Dong Wan Kim, Doe Hyun Yoon, Larry Kaplan, and Mattan Erez.** 2012. ACM/IEEE Supercomputing Conference.

18. *AutomaDeD: Automata-based Debugging for Dissimilar Parallel Tasks.* **Greg Bronevetsky, Ignacio Laguna, Saurabh Bagchi, Bronis R. de Supinski, Dong H. Ahn, Martin Schulz.** 2010. International Conference on Dependable Systems and Networks.

19. *Automatic Fault Characterization via Abnormality-Enhanced Classification.* **Greg Bronevetsky, Ignacio Laguna, Saurabh Bagchi and Bronis R. de Supinski.** 2012. International Conference on Dependable Systems and Networks (DSN).

20. *Introduction to Communication-Avoiding Algorithms.* **Demmel, Jim.** 2012. ACM/IEEE Supercomputing Conference - Tutorials.

21. *Comprehensive Algorithmic Resilience for Numeric Applications.* **Sui Chen, Greg Bronevetsky, Lu Peng, Marc Casas-Guix.** 2013. In Submission.